week 2

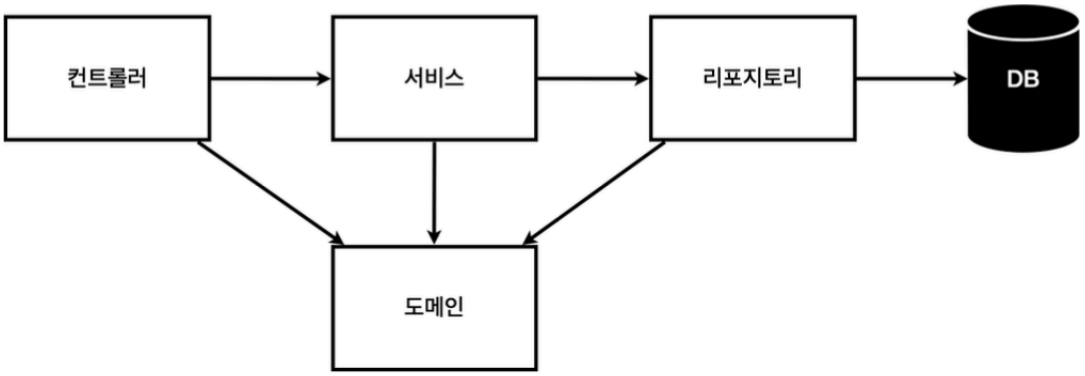
목표

간단한 비즈니스 로직을 구현하면서 스프링이 어떻게 동작하는지 알아보기

요구사항

- 데이터: 회원 ID, 이름
- 기능: 회원 등록, 조회
- 데이터 저장소가 선정되지 않음

웹 앱 계층 구조



- 컨트롤러: 웹 MVC의 컨트롤러
- 서비스: 핵심 비즈니스 로직 구현
- 리포지토리: 데이터베이스에 접근, 도메인 객체를 DB에 저장하고 관리
- 도메인: 비스니스 도메인 객체, 예) 회원, 주문, 쿠폰 등등 주로 DB에 저장하고 관리됨

Repository

먼저 간단한 데이터 저장소를 만들고 추후에 변경하기 위해
저장, 검색을 추상 메서드를 담은 인터페이스 작성

메모리 저장소

- HashMap<long, Member> 형태로 메모리에 저장

동시성 문제

여러 프로세스나 스레드에서 동시에 공유자원을 읽거나 수정하여 예상치 못한 잘못된 값을 얻는 경우가 있다.
위와 같은 상황이 동시성 문제이고, 문제가 생기는 것을 막기 위해 한 번에 하나의 연산만 수행하거나
하나의 프로세스/스레드만 접근할 수 있도록 락을 걸거나 synchronized 키워드를 사용한다.

저장소 테스트 케이스 작성

```
class ExampleTest {

    // junit으로 테스트 함수를 추가할 때는 @Test 어노테이션을 함수 위에 작성하기
    @Test
    public void test() {
        return ;
    }
```

테스트 클래스는 `src/test/java` 하위 디렉토리에 생성

정상적으로 동작하는 로직만 작성하지 말고, 예외 상황에서도 의도한 결과가 나오는지 확인

Test 어노테이션

https://junit.org/junit4/javadoc/4.12/org/junit/Test.html

```
@Test
public void test() {
    //do something
}
```

JUnit에게 아래의 `public void` 메서드가 테스트 케이스로 실행할 수 있다고 알려주는 역할

테스트 중에 어느 예외든 예외가 발생했다면 JUnit은 해당 테스트가 실패했다고 간주

`expected`, `timeout` 파라미터를 받을 수 있음

AfterEach 어노테이션

https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/AfterEach.html

```
@AfterEach
public void afterEach() {
    // do something
}
```

현재 메서드가 테스트 메서드가 실행되고 난 후 호출되어야 한다고 표시해주는 역할

aftereach 메서드는 `void` 타입이어야 하고, `private` , `static` 은 금지

`ParameterResolvers` 로 파라미터를 받을 수 있음

BeforeEach 어노테이션

https://junit.org/junit5/docs/5.0.2/api/org/junit/jupiter/api/BeforeEach.html

```
@BeforeEach
public void BeforeEach() {
    // do something
}
```

현재 메서드가 테스트 메서드가 실행되기 전에 호출되어야 한다고 표시해주는 역할

AfterEach와 같이 `void` 이어야 하고, `private` , `static` 금지,

`ParameterResolvers` 로 파라미터를 받을 수 있음

Component Scan

스프링에서 config 클래스 없이 자동으로 스프링 빈을 등록하는 컴포넌트 스캔 기능을 제공

@Componentscan

`@Component` , `@Service` , `@Repository` , `@Controller` 어노테이션이 붙은 클래스들을 자동으로 스캔하여 빈으로 등록

`@Configuration` 어노테이션이 붙은 클래스에 사용

현재 클래스의 패키지와 하위 패키지들을 모두 순회하며 빈 등록

Singleton pattern

클래스의 인스턴스 개수를 하나로 제한하는 디자인 패턴

```
public class Singleton {
    private static Singleton s = null;

    private Singleton getInstance() {
        if (s == null) {
            s = new Singleton();
        }
        return s;
    }
}
```

Dependency Injection

개발자가 정의한 빈 등록 정보를 바탕으로 스프링 컨테이너가 객체의 의존관계를 주입해주는 기능

의존성 주입을 통해 직접적인 결합 관계를 피하고 유지보수성을 높일 수 있음

Spring bean

스프링 IOC 컨테이너가 관리하는 객체

https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html

Constructor-based Dependency Injection

```
public class ConstructorBased {
    private final Dependency dp;

    // @Autowired
    // 생성자가 하나라면 생략 가능
    public ConstructorBased(Dependency dp) {
        this.dp = dp;
    }
}
```

Setter-Based Dependency Injection

```
public class SetterBased {
    private Dependency dp;

    @Autowired
    public void setDependency(Dependency dp) {
        this.dp = dp;
    }
}
```

```
public class MultipleArguments {
    private Dependency dp;
    private Dependency2 dp2;

    @Autowired
    public void prepare(Dependency dp, Dependency2 dp2) {
        this.dp = dp;
        this.dp2 = dp2;
    }
}
```

`@Autowired` 어노테이션을 생성자, setter, 메서드, 필드(권장 x) 에 적용하여 DI할 수 있음

`@Autowired`를 통한 DI는 스프링 빈으로 등록된 객체에서만 동작

form

HTML의 `<form>` 태그는 텍스트, 암호, 확인란 등 여러 데이터를 포함하는 양식을 서버로 전송하기 위한 태그

제출 양식은 다른 HTML 태그를 통해 정의

```
<!-- login form -->
<form action="/example/login" method="post">
    <input type="text" name="userName" placeholder="Email">
    <input type="password" name="userPassword" placeholder="Password">
    <label for="remember-check">
        <input type="checkbox" id="remember-check">아이디 저장하기
    </label>
```

```
    <input type="submit" value="Login">
</form>
```

입력 태그의 name 속성은 데이터를 받을 객체의 필드 이름과 동일하게 작성

DataSource

db 커넥션을 획득하기 위해 사용하는 객체

Connection pool이라는 캐시에 db 커넥션을 담아두고, 필요할 때 꺼내 사용하여 애플리케이션↔db 간의 연결 시간을 줄이는 역할

DataSource도 인터페이스이기 때문에 동작하기 위해서는 구현체가 있어야 함.

스프링 부트에서는 기본적으로 HikariDataSource를 사용

```
2025-02-07T15:52:40.681+09:00  INFO 20014 --- [hello-spring] [          main]
2025-02-07T15:52:40.757+09:00  INFO 20014 --- [hello-spring] [          main]
2025-02-07T15:52:40.758+09:00  INFO 20014 --- [hello-spring] [          main]
```

Spring 통합 테스트

@SpringBootTest

통합 테스트를 하려는 클래스에 사용

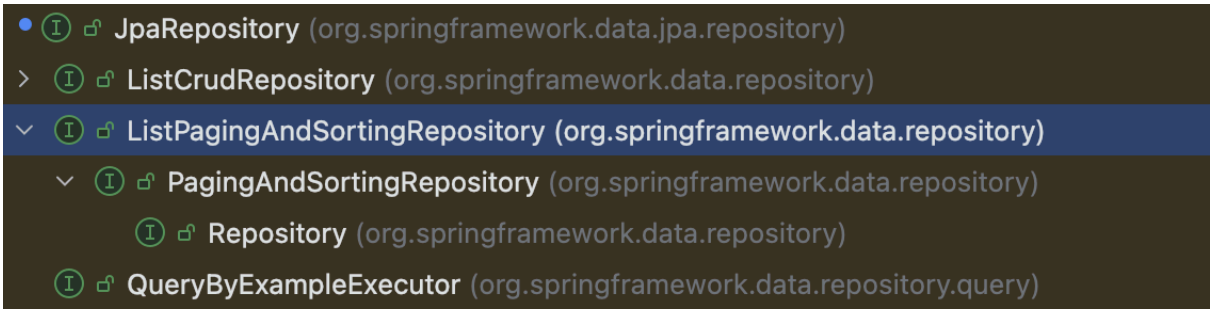
모든 빈들을 스캔하고 애플리케이션 컨텍스트를 생성하여 테스트

@Transactional

클래스 단위 혹은 메서드 단위로 선언

테스트 케이스에서 사용하면 변경사항들이 실제 db에 적용되지 않고 버려짐

서비스 클래스 같이 테스트 케이스가 아닌 곳에서 사용할 때는
CheckedException이거나 예외가 발생하지 않았을 때 db에 적용되고,
UncheckedException이 발생하면 db 롤백



JpaRepository의 페이징 기능은 `PagingAndSortingRepository` 에서 제공되는 기능